P01-1172/112056-0015

UNITED STATES PATENT APPLICATION

of

Peter F. Corbett

for a

PARITY ASSIGNMENT TECHNIQUE FOR PARITY DECLUSTERING IN A PARITY ARRAY OF A STORAGE SYSTEM

CROSS-REFERENCE TO RELATED APPLICATION

The present invention is related to the following co-pending and commonly assigned U.S. Patent Application Serial No. (112056-0016) titled, *Concentrated Parity Technique for Handling Double Failures and Enabling Storage of More than One Parity Block per Stripe on a Storage Device of a Storage Array*, which was filed on even date herewith and which application is hereby incorporated by reference as though fully set forth herein.

FIELD OF THE INVENTION

The present invention relates to arrays of storage systems and, more specifically, to a technique for efficiently reconstructing any one or combination of two failing storage devices of a storage array.

BACKGROUND OF THE INVENTION

A file server is a computer that provides file service relating to the organization of information on writeable persistent storage devices, such memories, tapes or disks of an array. The filer server or *filer* may be embodied as a storage system including a storage operating system that implements a file system to logically organize the information as a hierarchical structure of directories and files on the disks. Each "on-disk" file may be implemented as set of data structures, e.g., disk blocks, configured to store information, such as the actual data for the file. A directory, on the other hand, may be implemented as a specially formatted file in which information about other files and directories are stored.

A storage system may be further configured to operate according to a client/server model of information delivery to thereby allow many clients to access files stored on a server, e.g., the storage system. In this model, the client may comprise an application executing on a computer that "connects" to the storage system over a computer network, such as a point-to-point link, shared local area network, wide area network or virtual private network implemented over a public network, such as the Internet. Each client may request the services of the file system on the storage system by issuing file system protocol messages (in the form of packets) to the system over the network. It should be noted, however, that the storage system may alternatively be configured to operate as an assembly of storage devices that is directly-attached to a (e.g., client or "host") computer. Here, a user may request the services of the file system to access (i.e., read and/or write) data from/to the storage devices.

A common type of file system is a "write in-place" file system, an example of which is the conventional Berkeley fast file system. In a write in-place file system, the locations of the data structures, such as data blocks, on disk are typically fixed. Changes to the data blocks are made "in-place" in accordance with the write in-place file system. If an update to a file extends the quantity of data for the file, an additional data block is allocated.

Another type of file system is a write-anywhere file system that does not overwrite data on disks. If a data block on disk is retrieved (read) from disk into memory and "dirtied" with new data, the data block is stored (written) to a new location on disk to thereby optimize write performance. A write-anywhere file system may initially assume an optimal layout such that the data is substantially contiguously arranged on disks. The optimal disk layout results in efficient access operations, particularly for sequential read operations, directed to the disks. An example of a write-anywhere file system that is configured to operate on a storage system, such as a filer, is the Write Anywhere File Layout (WAFL™) file system available from Network Appliance, Inc., Sunnyvale, California. The WAFL file system is implemented as a microkernel within an overall protocol stack of the filer and associated disk storage.

The disk storage is typically implemented as one or more storage "volumes" that comprise a cluster of physical storage disks, defining an overall logical arrangement of disk space. Each volume is generally associated with its own file system. The disks within a volume/file system are typically organized as one or more groups of Redundant Array of Independent (or *Inexpensive*) Disks (RAID). RAID implementations enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of redundant information with respect to the striped data. The redundant information enables recovery of data lost when a storage device fails.

In the operation of a disk array, it is fairly common that a disk will fail. A goal of a high performance storage system is to make the mean time to data loss (MTTDL) as long as possible, preferably much longer than the expected service life of the system. Data can be lost when one or more storage devices fail, making it impossible to recover data from the device. Typical schemes to avoid loss of data include mirroring, backup and parity protection. Mirroring is an expensive solution in terms of consumption of storage resources, such as hard disk drives. Backup does not protect recently modified data. Parity schemes are common because they provide a redundant encoding of the data that allows for a single erasure (loss of one disk) with the addition of just one disk drive to the system.

Parity protection is used in computer systems to protect against loss of data on a storage device, such as a disk. A parity value may be computed by summing (usually modulo 2) data of a particular word size (usually one bit) across a number of similar disks holding different data and then storing the results on an additional similar disk. That is, parity may be computed on vectors 1-bit wide, composed of bits in corresponding positions on each of the disks. When computed on vectors 1-bit wide, the parity can be either the computed sum or its complement; these are referred to as even and odd parity respectively. Addition and subtraction are on 1-bit vectors equivalent to an exclusive-OR (XOR) logical operation, and the addition and subtraction operations are replaced by XOR operations. The data is then protected against the loss of any of the disks. If the disk storing the parity is lost, the parity can be regenerated from the data. If one of the data disks is lost, the data can be regenerated by adding the contents of the surviving data disks together and then subtracting the result from the stored parity.

Typically, the disks are divided into parity groups, each of which comprises one or more data disks and a parity disk. A parity set is a set of blocks, including several data and one parity block, where the parity block is the XOR of all the data blocks. A parity group is a set of disks from which one or more parity sets are selected. The disk space is divided into stripes, with each stripe containing one block from each disk. The blocks of a stripe are usually at the same locations on each disk in the parity group. Within a stripe, all but one block are blocks containing data ("data blocks") and one block is a block containing parity ("parity block") computed by the XOR of all the data. If the parity blocks are all stored on one disk, thereby providing a single disk that contains all (and only) parity information, a RAID-4 implementation is provided. If the parity blocks are contained within different disks in each stripe, usually in a rotating pattern, then the implementation is RAID-5. The term "RAID" and its various implementations are well-known and disclosed in *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, by D. A. Patterson, G. A. Gibson and R. H. Katz, Proceedings of the International Conference on Management of Data (SIGMOD), June 1988.

As used herein, the term "encoding" means the computation of a redundancy value over a predetermined subset of data blocks, whereas the term "decoding" means the reconstruction of a data or parity block by the same or similar process as the redundancy computation using a subset of data blocks and redundancy values. If one disk fails in the parity group, the contents of that disk can be decoded (reconstructed) on a spare disk or disks by adding all the contents of the remaining data blocks and subtracting the result from the parity block. Since two's complement addition and subtraction over 1-bit fields are both equivalent to XOR operations, this reconstruction consists of the XOR of all the surviving data and parity blocks. Similarly, if the parity disk is lost, it can be recomputed in the same way from the surviving data.

An aspect of parity protection of data is that it provides protection against only a single disk failure within a parity group. These parity schemes can also protect against multiple disk failures as long as each failure occurs within a different parity group. However, in the case of multiple simultaneous failures within a parity group, no such protection is provided and an unrecoverable loss of data is suffered. Failure of two disks concurrently within a parity group is a fairly common occurrence, particularly because disks "wear out" and because of environmental factors with respect to the operation of the disks. In this context, the failure of two disks concurrently within a parity group is referred to as a "double failure".

A double failure typically arises as a result of a failure to one disk and a subsequent failure to another disk while attempting to recover from the first failure. The recovery or reconstruction time is dependent upon the level of activity of the storage system. That is, during reconstruction of a failed disk, it is desirable that the storage system remain "online" and continue to serve requests (from clients or users) to access (i.e., read and/or write) data. If the storage system is busy serving requests, the elapsed time for reconstruction increases. The reconstruction processing time also increases as the number of disks in the storage system increases. Moreover, the double disk failure rate is proportional to the square of the number of disks in a parity group. However, having small parity groups is expensive, as each parity group requires an entire disk devoted to redundant data.

Accordingly, it is desirable to provide a technique that withstands double failures. This would allow construction of larger disk systems with larger parity groups, while ensuring that even if reconstruction after a single disk failure takes a long time (e.g., a number of hours), the system can survive a second failure. Such a technique would further allow relaxation of certain design constraints on the storage system. For example, the storage system could use lower cost disks and still maintain a high MTTDL. Lower cost disks typically have a shorter lifetime, and possibly a higher failure rate during their lifetime, than higher cost disks. Therefore, use of such disks is more acceptable only if the system can withstand double disk failures within a parity group.

A known scheme for protecting against double disk failures is a distributed parity scheme disclosed in U.S. Patent No. 5,862,158, titled *Efficient Method for Providing Fault Tolerance Against Double Device Failures in Multiple Device Systems*, by Baylor et al., issued January 19, 1999, which patent is hereby incorporated by reference as though fully set forth herein. This distributed parity scheme, hereinafter referred to as the "Corbett-Park" scheme, provides a way of determining the assignment of each data block to exactly two parity sets, such that all parity sets are the same size. The scheme also allows recovery from any two disk failures using an optimal amount of parity information, equivalent to two disks worth, for any even number of disks 4 or greater, except 8.

The Corbett-Park technique allows a tradeoff between the optimal amount of parity and the number of data blocks that belong to each parity set, i.e., the number of data blocks XOR'd to compute each parity block. For a given number of disks n in the array, a ratio m may be selected, which is the number of data blocks per parity block. Thus, the redundancy is $1/(m+1)$ and the parity overhead is $1/m$, wherein m is restricted such that $2m+2 \leq n$. Within each recurrence of the parity assignment pattern, each disk has m data blocks and one parity block. Without losing generality, assume that the data blocks are in m rows and the parity blocks are in a single row. A stripe is composed of $m+1$ rows. Therefore, there is a minimum of $n/(m+1) = (2m+2)/(m+1) = 2$ disks worth of parity, which is the optimal amount. If n is made larger than $2m+2$, then there is more than the minimum parity in the parity array, but the overhead and redundancy may be maintained by keeping m constant.

It should be noted that the sizes of the blocks used in the parity computations for the conventional Corbett-Park scheme do not have to match the sizes of any system blocks. Therefore, the blocks used for parity computation can be chosen to be some convenient size. For example, assume it is desired that each recurrence of the parity assignment pattern contain exactly 4 kilobytes (kB) of data from each disk. The sizes of the blocks used for parity computation can thus be defined as 4k/(m+1), which works well if $m = 3$, 7 or 15 when the block size is a larger power of two, such as 4k (4096) bytes.

The result of the Corbett-Park parity assignment construction technique is that within each row of a stripe, the data block on disk i belongs to parity sets $(i+j) \bmod n$ and $(i+k) \bmod n$, where $j \neq k$, $0<j<n$ and $0<k<n$. The parity block for parity set p is stored on disk p, wherein $0 \leq p<n$. The difference between values j and k is a parity delta, wherein a parity delta is the difference in value modulo the number of disks between two parity sets to which a data block belongs. Note that this form of modulo arithmetic is preferably "wrap around" addition, which is similar to discarding carries in an XOR operation.

Specifically, parity offset is defined as, in a row, the differences j and k between a disk number d and the parity sets to which a block belongs, such that set1 $= (d + j) \bmod n$

and set2 = $(d + k) \bmod n$. Here, j and k are the parity offsets. Parity offsets are between 1 and n - 1, inclusive. Parity delta, on the other hand, is defined as the difference, modulo n, between the parity offsets j and k for a row of blocks. The parity deltas for a row are $(j - k) \bmod n$ and $(k - j) \bmod n$. Parity deltas are between 1 and n -1, inclusive, and $n/2$ is not allowed, as are other factors of n depending on the depth of the parity blocks per stripe in the relocated parity arrangement.

Among all the m rows of data blocks, all the deltas must be unique. Furthermore, in cases of even values of n, delta of $n/2$ is not allowed. Therefore, for m rows, there are 2 x m unique deltas required. Given n-1 possible deltas in the case of odd n, and n-2 possible deltas in the case of even n, there is a restriction on the size of m and, hence, on the minimum redundancy ratio for a given number of disks. The assignment of two unique parity deltas to each row of a stripe as described herein ensures five conditions:

1. Each data block belongs to two unique parity sets.
2. The m data blocks on each disk belong to 2 x m different parity sets.
3. No data block on a given disk belongs to the parity set of the parity block on that disk.
4. All parity sets are of the same size. Since each data block belongs to exactly the minimum number 2 parity sets, a corollary is that all parity sets are of minimum size.
5. On each disk, there is at least one parity set that is not represented by either a data or parity block.

These conditions are necessary, but not sufficient to ensure that the array contents can be reconstructed after two device failures. The 5th condition is key to reconstruction. When two disks are lost, there are always at least two parity sets for which only one block is lost and which allow for reconstruction of the missing block. If $n = 2m + 2$, there are exactly two parity sets that are only missing one member. If $n > 2m + 2$, there are more than two parity sets that are only missing one member and, in fact, some parity sets may be complete and require no reconstruction. Therefore reconstruction can always begin by reconstructing two of the missing blocks, one from each of the failed disks. If the reconstructed blocks are data blocks, then each reconstructed data block allows reconstruction of the missing block from another parity set, since each data block belongs to two parity sets. Each reconstruction "chain" is continued until a parity block is reconstructed.

Fig. 1 is a block diagram of a prior art 4-disk array 100 configured in accordance with the Corbett-Park distributed parity arrangement. Each disk is assigned parity and data blocks that are organized into stripes and assigned to parity sets. For example, each stripe contains two blocks, a data (D) block and a parity (P) block, with the parity and data block assignment patterns repeated for each stripe. Each parity block stores the parity for one parity set and each parity set has exactly one parity block. All parity sets are of equal size and are composed, in addition to the parity block, of several data blocks. The parity set size is thus equal to the number of data blocks and their associated parity block. Each data block is defined to be a member of two parity sets. No two data blocks in the array are both members of the same two parity sets. No more than one data block or parity block on any given disk belongs to any one parity set.

In the case of any single failure, no parity set misses more than one block and therefore all parity sets, and consequently all missing blocks, can be reconstructed. If any one of the disks fails, the lost data and parity blocks can be reconstructed, since at most one block is missing from all parity sets. Moreover, if any two disks fail, all the missing blocks can be generated from the remaining parity and data on the other two disks.

For example, if disks 0 and 1 fail, parity sets 1 and 2 are each missing two blocks and cannot be reconstructed immediately. However, each parity set 0 and 3 is missing only one block and therefore the missing blocks of these parity sets can be reconstructed. Reconstructing data block D23 (the missing data block from parity set 3) results in the reconstruction of a data block member of parity set 2 which, in turn, allows reconstruction of data block D12 on disk 0. By reconstructing data block D12, the parity block of parity set 1 on disk 1 can be reconstructed. Note that the parity blocks end the reconstruction chains. That is, for a double failure (two disk failures) there are always two reconstruction chains; recovery is effected by reconstructing the data blocks prior to the parity blocks of those chains, and then reconstructing the two parity blocks of the two failed disks.

Assume now that disks 0 and 2 fail. Each parity set 0 and 2 is missing two blocks, while each parity set 1 and 3 is missing only one block. Therefore, the missing blocks of parity sets 1and 3 can be reconstructed immediately. In particular, the missing data block D12 from parity set 1 can be reconstructed which enables reconstruction of parity block 2 on disk 2. Similarly, missing data block D30 from parity set 3 can be reconstructed, which enables reconstruction of parity block 0 on disk 0. Therefore, there are again two chains that are reconstructed starting with the missing data block and ending with the parity block.

Fig. 2 is a block diagram of a prior art 6-disk array 200 configured in accordance with the Corbett-Park distributed parity arrangement. Each disk is divided into data and parity blocks that are further organized into stripes. Each stripe contains two data blocks and one parity block, and there is an optimal amount of parity information equivalent to two disks worth. Each data block belongs to two parity sets and there is only one (at most) representative member of each parity set on each disk. Notably, each disk is missing one parity set and the missing parity set is different among all the disks. Furthermore, each row of blocks within each stripe has a different parity delta. For example, row 1 has a parity delta of Δ1 and row 2 has a parity delta of Δ2. The same delta is not allowed on two different rows because it renders the system vulnerable to failures that distance apart.

Specifically, the Corbett-Park construction technique specifies that every row of data blocks in the parity array has a different parity delta value and that no row's delta value equals *n*/2. For example, in a 6-disk array, no row has a delta value of 3 (Δ3); similarly in a 10-disk array, no row has a delta value of 5 (Δ5). A parity delta cannot have a value of *n*/2 because there cannot be a "harmonic" of parity set assignments to a data block within an array. It should be noted, however, that each row essentially has a pair of parity delta values. That is, in the case of a 10-disk array, a delta value of Δ3 is equivalent to a delta value of Δ7, and a delta value of Δ4 is equivalent to a delta value of Δ6. Similarly, in a 6-disk array, a delta value of Δ1 is equivalent to a delta value of Δ5 and a delta value of Δ2 is equivalent to a delta value of Δ4.

The Corbett-Park construction technique also requires finding a vector of n elements containing two each of the symbols from 1 to m that represent the data blocks, one symbol for parity (e.g., -1) and another set of symbols for null values (e.g., 0). The object is to find an n element vector that allows a complete marking of all symbols in the vector and all super-positions of the vector on itself. A vector that successfully meets the criteria corresponds to a solution of the parity assignment problem for a given n and m that allows recovery from any double disk failure.

Fig. 3 is a table 300 illustrating construction of parity assignments in a parity array from a successful vector in accordance with the Corbett-Park construction technique. The vector is {-1, 1, 0, 2, 2, 1}, which is one of four unique solutions for $n = 6$, $m = 2$ (discounting congruent solutions). Each column of the table represents a disk and each row represents a parity set. The positions of the symbols in each column indicate the assignment of the data blocks 1 and 2, along with the parity block –1, to a parity set in that disk. Symbol 1 represents the data blocks in row 1 and the symbol 2 represents the data blocks in row 2 of a stripe. The data and parity blocks may be arranged in any order within each disk. The parity assignment vector is rotated by one position in each disk ensuring that it occupies all n possible positions in the n disk array. Fig. 4 is a block diagram illustrating the parity assignment pattern resulting from the construction technique.

As noted, the Corbett-Park technique provides a means for assigning data blocks to parity sets in a distributed parity configuration having an optimal amount or sub-optimal amount of parity and a chosen redundancy ratio. That is, for a given number of disks n, an optimal amount of parity is chosen to be equal to $2/n$. However for odd numbers of disks n there are solutions that are sub-optimal (i.e., more than two disks worth of parity) but that operate correctly. For example, if $n = 7$ and each disk is divided into three blocks (two data blocks and one parity block), there is a total of 7/3 of parity which is greater than two disks worth and thus a sub-optimal solution resulting in increased redundancy. Increasing redundancy has two effects: (1) the size of the parity sets is smaller than the maximum allowed for a given number of disks, thus effectively "declustering" the parity. As n increases, there are more parity sets that are not affected at all by the failures. Particularly in the more common case of single failures, this might result in fewer disk read operations during reconstruction. (2) There are more than two parity sets that only lose one block in the event of the failures, increasing the parallelism possible during reconstruction.

Typically, in a RAID-4 or RAID-5 implementation of a disk array, when a disk fails (is lost) data stored on all surviving disks must be read (retrieved) in order to reconstruct the lost disk, and, in particular, the lost data. Retrieving data from all of the surviving disks is expensive in terms of the time required to access data from those disks. This is particularly true for a storage system (e.g., functioning as a server) configured to continue serving (client) read and write requests during reconstruction of the failed disk.

Accordingly, parity declustering is used to distribute the workload during reconstruction, particularly for single disk failure but also for double failures. The notion of parity declustering generally involves distributing the association of data blocks to parity sets as uniformly as possible across the array to thereby distribute the workload when reconstructing the failed disk(s). Parity declustering is discussed in Holland M., Gibson G. *Parity Declustering for Continuous Operation in Redundant Disk Arrays*, Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), October 1992, Boston, MA, ACM Press, pp 23-35, which article is hereby incorporated by reference as though fully set forth herein.

The present invention is directed to a technique that enables parity declustering with respect to a concentrated parity technique. In addition, the present invention is directed to a modification of the conventional Corbett-Park technique that allows for parity declustering in large disk arrays, particularly for large balanced arrays.

SUMMARY OF THE INVENTION

The present invention comprises a parity assignment technique that enables parity declustering in a large, balanced parity ("super-stripe") array of a storage system. The balanced array may be constructed by combining a plurality of unbalanced parity stripe arrays, each having parity blocks on a set of storage devices, such as disks, that are disjoint from the set of disks storing the data blocks. The novel parity assignment technique distributes the assignment of disks to parity groups throughout the combined super-stripe

array such that all disks contain the same amount of data or parity information. Moreover, the inventive technique ensures that all surviving data disks of the array are loaded uniformly during a reconstruction phase after a single or double disk failure.

According to the present invention, parity declustering can be achieved by selecting patterns of characters, such as binary numbers, representing the data block disks constituting a stripe to thereby change the association of the data disks with parity groups from stripe to stripe of the balanced super-stripe array. In this context, parity declustering refers to distributing the association of data disks to parity groups more widely than initially configured so as to distribute the reconstruction workload across all surviving disks as much as possible. In other words, the parity assignment technique changes the parity group association of data blocks of a data disk with respect to other data blocks of data disks of the array. Thereafter, in the event of a single or double disk failure, the reconstruction load may be distributed across all of the surviving data disks.

Advantageously, the present invention ensures that during reconstruction of the failed disk(s), only a fraction of the data from all of the data disks is retrieved. That is, instead of accessing all data from some of the data disks, and accessing none of the data from others of those disks, the parity assignment technique ensures that all data disks are partially (i.e., not fully) accessed. Notably, the inventive technique applies to both distributed and concentrated parity techniques, particularly where the array consists of balanced super-stripes that are tolerant of any two disk failures.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

Fig. 1 is a block diagram of a first, prior art disk array configured in accordance with a conventional distributed parity arrangement;

Fig. 2 is a block diagram of a second, prior art disk array configured in accordance with the conventional distributed parity arrangement;

Fig. 3 is a table illustrating construction of parity assignments in a parity array from a vector in accordance with a prior art construction technique;

Fig. 4 is a block diagram illustrating a parity assignment pattern resulting from the prior art construction technique;

Fig. 5 is a schematic block diagram of an environment including a file server that may be advantageously used with the present invention;

Fig. 6 is a schematic block diagram of a storage operating system including a write anywhere file layout (WAFL) file system layer that may be advantageously used with the present invention;

Fig. 7 is a block diagram of data and parity blocks of a balanced stripe array according to a concentrated parity technique that may be advantageously used with the present invention;

Fig. 8 is a block diagram illustrating parity assignments for a disk array;

Fig. 9 is a block diagram illustrating a balanced parity super-stripe array that may be advantageously used with the invention;

Fig. 10 is a block diagram illustrating a balanced parity super stripe array configuration adapted for large write operations that may be advantageously used with the present invention;

Fig. 11 is a block diagram illustrating a parity assignment technique of the present invention as applied to a large balanced array of Figs. 9 and 10;

Fig. 12 is a diagram illustrating the assignment of data disks to parity groups in a repeating pattern according to the present invention;

Fig. 13 is a diagram illustrating an assignment pattern of data disks to parity groups using binary digits;

Fig. 14 is a diagram of an array having a plurality of data disks assigned to parity groups using binary digits in accordance with a parity assignment pattern using the inventive technique; and

Fig. 15 is a diagram showing a partial parity assignment pattern for a balanced array comprising three arrays, each having two disks.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

Fig. 5 is a schematic block diagram of an environment 500 including a file server, such as a network storage appliance, that may be advantageously used with the present invention. The file server or filer 520 is a computer that provides file service relating to the organization of information on storage devices, such as disks 530 of a disk array 560. The filer 520 comprises a processor 522, a memory 524, a network adapter 526 and a storage adapter 528 interconnected by a system bus 525. The filer 520 also includes a storage operating system 600 that implements a file system to logically organize the information as a hierarchical structure of directories and files on the disks.

In the illustrative embodiment, the memory 524 comprises storage locations that are addressable by the processor and adapters for storing software program code and data structures associated with the present invention. The processor and adapters may, in turn, comprise processing elements and/or logic circuitry configured to execute the software code and manipulate the data structures. The storage operating system 600, portions of which are typically resident in memory and executed by the processing elements, functionally organizes the filer by, *inter alia*, invoking storage operations in support of a file service implemented by the filer. It will be apparent to those skilled in the art that other processing and memory means, including various computer readable media, may be used for storing and executing program instructions pertaining to the inventive technique described herein.

The network adapter 526 comprises the mechanical, electrical and signaling circuitry needed to connect the filer 520 to a client 510 over a computer network 540, which may comprise a point-to-point connection or a shared medium, such as a local area network. The client 510 may be a general-purpose computer configured to execute applications 512. Moreover, the client 510 may interact with the filer 520 in accordance with a client/server model of information delivery. That is, the client may request the services of the filer, and the filer may return the results of the services requested by the client, by exchanging packets 550 encapsulating, e.g., the Common Internet File System (CIFS) protocol or Network File System (NFS) protocol format over the network 540.

The storage adapter 528 cooperates with the storage operating system 600 executing on the filer to access information requested by the client. The information may be stored on any type of attached array of writeable media such as video tape, optical, DVD, magnetic tape, bubble memory and any other similar media adapted to store information, including data and parity information. In the illustrative embodiment described herein, however, the information is preferably stored on the disks 530 of array 560. The storage adapter includes input/output (I/O) interface circuitry that couples to the disks over an I/O interconnect arrangement, such as a conventional high-performance, Fibre Channel serial link topology. The information is retrieved by the storage adapter and, if necessary, processed by the processor 522 (or the adapter 528 itself) prior to being forwarded over the system bus 525 to the network adapter 526, where the information is formatted into a packet and returned to the client 510.

Storage of information on array 560 is preferably implemented as one or more storage "volumes" that comprise a cluster of physical storage disks 530, defining an overall logical arrangement of disk space. Each volume is generally associated with its own file system. The disks within a volume/file system are typically organized as one or more groups of Redundant Array of Independent (or *Inexpensive*) Disks (RAID). RAID implementations enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of parity information with respect to the striped data.

To facilitate access to the disks 530, the storage operating system 600 implements a write-anywhere file system that logically organizes the information as a hierarchical structure of directories and files on the disks. Each "on-disk" file may be implemented as a set of disk blocks configured to store information, such as data, whereas the directory may be implemented as a specially formatted file in which other files and directories are stored. In the illustrative embodiment described herein, the storage operating system is preferably the NetApp® Data ONTAP™ operating system available from Network Appliance, Inc., Sunnyvale, California that implements a Write Anywhere File Layout (WAFL™) file system. It is expressly contemplated that any appropriate file system can be used, and as such, where the term "WAFL" is employed, it should be taken broadly to refer to any file system, database or data storage system that is otherwise adaptable to the teachings of this invention.

Fig. 6 is a schematic block diagram of the Data ONTAP operating system 600 that may be advantageously used with the present invention. The storage operating system comprises a series of software layers, including a media access layer 610 of network drivers (e.g., an Ethernet driver). The operating system further includes network protocol layers, such as the Internet Protocol (IP) layer 612 and its supporting transport mechanisms, the Transport Control Protocol (TCP) layer 614 and the User Datagram Protocol (UDP) layer 616. A file system protocol layer provides multi-protocol data access and, to that end, includes support for the CIFS protocol 618, the NFS protocol 620 and the Hypertext Transfer Protocol (HTTP) protocol 622. In addition, the storage operating system 600 includes a disk storage layer 624 that implements a disk storage protocol, such as a RAID protocol, and a disk driver layer 626 that implements a disk access protocol such as, e.g., a Small Computer Systems Interface (SCSI) protocol.

Bridging the disk software layers with the network and file system protocol layers is a WAFL layer 680 that preferably implements the WAFL file system. The on-disk format representation of the WAFL file system is block-based using, e.g., 4 kilobyte (kB) blocks and using inodes to describe the files. The WAFL file system uses files to store meta-data describing the layout of its file system; these meta-data files include, among others, an inode file. A file handle, i.e., an identifier that includes an inode number, is used to retrieve an inode from disk.

Operationally, a request from the client 510 is forwarded as, e.g., a conventional CIFS or NFS protocol packet 550 over the computer network 540 and onto the filer 520 where it is received at the network adapter 526. A network driver of the media access layer 610 processes the packet, passes it onto the network protocol layers 612-616 and CIFS or NFS layer 618, 620 for additional processing prior to forwarding to the WAFL layer 680. Here, the WAFL file system generates operations to load (retrieve) the requested data from disk 530 if it is not resident "incore", i.e., in the memory 524. If the information is not in memory, the WAFL layer 680 indexes into the inode file using the inode number to access an appropriate entry and retrieve a logical volume block number

(VBN). The WAFL layer then passes the logical VBN to the disk storage (RAID) layer 624, which maps that logical number to a disk block number and sends the latter to an appropriate driver (e.g., SCSI) of the disk driver layer 626. The disk driver accesses the disk block number from disk 530 and loads the requested data block(s) in memory 524 for processing by the filer. Upon completion of the request, the filer (and operating system) returns a reply to the client 510 over the network 540.

It should be noted that the software "path" through the storage operating system layers described above needed to perform data storage access for the client request received at the filer may alternatively be implemented in hardware. That is, in an alternate embodiment of the invention, the storage access request data path 650 may be implemented as logic circuitry embodied within a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC). This type of hardware implementation increases the performance of the file service provided by filer 520 in response to a file system request packet 550 issued by client 510. Moreover, in another alternate embodiment of the invention, the processing elements of adapters 526, 528 may be configured to offload some or all of the packet processing and storage access operations, respectively, from processor 522, to thereby increase the performance of the file service provided by the filer. It is expressly contemplated that the various processes, architectures and procedures described herein can be implemented in hardware, firmware or software.

As used herein, the term "storage operating system" generally refers to the computer-executable code operable to perform a storage function in a storage system, e.g., that implements file system semantics and manages data access. In this sense, the ONTAP software is an example of such a storage operating system implemented as a microkernel and including the WAFL layer to implement the WAFL file system semantics and manage data access. The storage operating system can also be implemented as an application program operating over a general-purpose operating system, such as UNIX® or Windows NT®, or as a general-purpose operating system with configurable functionality, which is configured for storage applications as described herein.

In addition, it will be understood to those skilled in the art that the inventive technique described herein may apply to any type of special-purpose (e.g., server) or general-purpose computer, including a standalone computer, embodied as or including a storage system. To that end, filer 520 is hereinafter described as an exemplary implementation of a storage system 520. Moreover, the teachings of this invention can be adapted to a variety of storage system architectures including, but not limited to, a network-attached storage environment, a storage area network and disk assembly directly-attached to a client or host computer. The term "storage system" should therefore be taken broadly to include such arrangements in addition to any subsystem configured to perform a storage function and associated with other equipment or systems.

The present invention may operate in accordance with an illustrative concentrated parity technique that utilizes parity protection to protect against all single and double failures, where a single failure is a loss of one storage device, such as disk 530, and a double failure is a loss of two disks 530 of array 560. The concentrated parity technique is preferably implemented by the disk storage (RAID) layer 624 that assigns data blocks to parity sets in accordance with a conventional distributed parity technique. The RAID layer cooperates with the WAFL layer 680 to divide the disks 530 into data and parity blocks that are then organized as RAID groups through the redundant writing of stripes across the disks, wherein each stripe contains data and parity blocks from each of the disks. To that end, the WAFL layer generally performs large write operations, writing or overwriting an entire stripe (i.e., the blocks of one or more overlapping parity sets) of a parity array at once. The contents of the stripe, including the parity blocks, are computed in memory before the stripe is written to disk.

In the illustrative embodiment of a WAFL-based file system and process, a RAID-4 implementation is advantageously employed. This implementation specifically entails the striping of data across a group of disks, and separate parity storage on selected disk(s) of the RAID group. The illustrative RAID-4 implementation of the disk array as described herein provides the flexibility to start with an under-populated disk array, and add disks individually or in small numbers until a predetermined, fully populated array is achieved. This is possible because an initial situation may be assumed wherein the array has a finite number of imaginary data disks, all containing zeros. With a single parity disk, data disks can be added until the size of a stripe becomes too large to buffer effectively.

In accordance with the illustrative RAID-4 style technique, all data is stored on a subset of disks in a parity group and all parity is stored on a disjoint subset of disks. This may be accomplished by first noting that all the rows of parity blocks may be removed from the original disks of the array and moved to the disjoint set of disks of the "extended" array. Such an extended array would still be tolerant of double failures because any two data disks failing would result in less loss of information than would have occurred if the parity blocks were still mixed with the data blocks. Any loss of a disk containing a parity block would similarly be easier to recover from than if both the parity and data from the original disk were lost.

The illustrative concentrated parity technique enables construction of an extended array of disks having a data block portion that is maintained on an original set of disks, while a parity block portion of the array is contained on a physically separate, i.e., disjoint, set of disks. Within the set of parity disks that is disjoint from the set of disks storing the data blocks, the invention enables storage of more than one parity block on each disk. Referring to the parity assignment pattern for the 6-disk array of Fig. 2, a redundancy ratio of 1/3 is provided, i.e., one third of the disk storage is occupied by parity. The disks employ a distributed parity configuration as indicated by one parity block and two data blocks per disk (per stripe). One way of achieving the concentrated parity arrangement is to take all the parity blocks P0-P5 and place them on six parity disks that are disjoint from the original six disks containing the data blocks. Here, the redundancy ratio of the novel arrangement remains the same. However the amount of parity is not optimal with respect to the total disk storage occupied by parity as 6 disks, instead of 2 disks, contain parity.

Although the concentrated parity configuration described herein is tolerant of double failures, it may create unbalanced disk lengths with respect to the blocks per disk per stripe. The parity blocks may be further combined and arranged such that all disks contain the same number of blocks (e.g., two) to thereby provide a balanced stripe "array" across the disks. Fig. 7 is a block diagram of data and parity blocks of a balanced stripe array 700 according to the concentrated parity technique. Here, parity blocks P0 and P3 are both stored within a disk, as are parity blocks P1 and P4, and parity blocks P2 and P5. This "self-contained" stripe of data and parity may be repeated in a recurring pattern across all the disks. Notably, this combination of parity blocks represents the only functional arrangement for the 6 data disk array system.

Specifically, the difference of the combined parity blocks on each parity disk is 3. A parity delta of Δ3 is prohibited for a row of data blocks because it results in a non-functional arrangement. A delta of Δ3 is non-functional (i.e., does not work) for a row of data blocks because it ascribes two data blocks having positions of exactly half the length of the array apart to exactly the same parity set assignments. If two disks half the length of the array apart were lost, then two data block members of the same two parity sets are lost, thereby obviating recovery of the lost data. Row 1 of the parity array thus has a delta of Δ1 and row 2 has a delta of Δ2.

The assignment prohibition of certain parity deltas to data blocks described above also applies to the conventional Corbett-Park distributed parity arrangement. However, it is this prohibition (restriction) that provides the basis of the illustrative concentrated parity variant of that conventional technique, as described herein. In other words, the conventional Corbett-Park construction technique specifies that for an even number of disks *n*, there is no delta of *n*/2. By excluding a parity delta value of *n*/2 from assignments to data blocks, certain parity blocks (i.e., those having a difference of *n*/2) may be combined on the same disk, since no data block can be in both of the parity sets separated by *n*/2.

For example in the case of a 6-disk system, the parity blocks can be removed from the original disks and combined on disjoint disks only if the parity block parity assignments are separated by 3, while the data block parity assignments are maintained as originally assigned. Thus, parity blocks having a delta of Δ3 may be combined on parity disks 6-8 of array 700. Note that when a parity disk is lost, two parity blocks from each stripe are lost. The two lost parity blocks should not belong to the two parity sets to which any disk block belongs. This aspect of the concentrated parity technique is guaranteed because no disk block is assigned to two parity sets that are separated by a delta of Δ3. This aspect is also notable for a write anywhere file system that utilizes a RAID-4 style concentrated parity arrangement. Yet, it should be noted that the concentrated parity arrangement may apply to any type of file system, including a write in-place file system, having any type of attached array of any type of writeable persistent storage media, such as video tape, optical, DVD, magnetic tape and bubble memories.

According to the concentrated parity technique, more than two parity blocks may be combined onto a single parity disk within each stripe. By precluding the assignment of certain deltas to data blocks that are a factor of n (where n = the total number of data disks in the array), the corresponding parity blocks may be grouped together. Ideally, it is desirable to combine the parity blocks into n/m disks with m parity blocks on each disk, as this would provide a match between the size of the data disks and the size of the parity disks. The recurring pattern would have m blocks on each disk, either data or parity. For certain factors f_1, f_2 of n, where $f_1 \times f_2 = n$, solutions are available that allow placement of f_1 parity blocks on f_2 dedicated parity disks. The restriction is that the pair of deltas in each row of data blocks must obey the following relationship, where i and j are the two parity deltas for a row:

$$(i - j) \bmod n \neq k f_2, \text{ for any value of } k \text{ such that } 0 < k f_2 < n$$

In other words, the restriction specifies no $\Delta = k f_2$ for a row of data blocks. A feature of the concentrated parity technique is that once values of n and m are established, it is necessary to find only one operational combination of parity deltas. That combination can be used in the encoding process in all arrays. It can be shown that the restriction described herein results in a configuration where no parity disk contains two parity blocks to which any one data block belongs.

For example, assume a 12-disk array with redundancy of 1 parity block for every 4 data blocks. This array can be configured as a RAID-5 style array according to the conventional Corbett-Park scheme with 4 data blocks and 1 parity block in each recurrence of the parity assignment pattern per disk. The parity blocks can be stripped from the 12-disk array and placed, 4 blocks per disk, on 3 additional parity disks, thereby creating a RAID-4 style array. This resulting 15-disk array includes 12 data disks and 3 parity disks. Although there is more than the minimum two disks of parity in the array, the ratio of parity to data is unchanged from the original array and the parity set size is unchanged. There are more disks but there is also more space for data on the original 12 disks. An advantage of this technique is that the array can be configured with a smaller number of data disks and, as additional data disks are needed, those disks can be added individually or in small groups without moving any data blocks or reconfiguring or re-computing any parity. The concentrated parity technique simply "assumes" that all 12 data disks are present from the beginning and that any disks that aren't actually present in the array are filled with, e.g., zero values.

An example of the restriction specified by the concentrated parity technique is as follows. Assume that a 15-disk array is configured as 12 data disks and 3 parity disks. Since $f_2 = 3$ and no $\Delta = k f_2$ is allowed, deltas having a value of $\Delta 3$, $\Delta 6$ or $\Delta 9$ are not allowed for assignment of data blocks to parity sets. Given that restriction, parity blocks P0, P3, P6 and P9 can be combined onto one parity disk, parity blocks P1, P4, P7, P10 can be combined onto another parity disk and parity blocks P2, P5, P8, P11 can be combined onto a third parity disk. In essence, the restriction specifies that, depending upon the number of parity disks (f_2) where f_2 is a factor of n, there cannot be any multiple of that number ($k f_2$) used as a parity delta for the data blocks. By precluding data block parity set assignments of multiples of the parity disk (e.g., 3, 6 and 9 for a 12-disk array), the concentrated parity technique allows combining of parity blocks having those differences, modulo n, on a single disk.

Fig. 8 is a table 800 illustrating a solution of parity assignments for a 15-disk array having 12 data disks and 3 parity disks, with a redundancy ratio of 4 to 1. Each row represents the contents of one row of the recurring pattern of parity assignment on disk. The numbers associated with each data and parity block are the parity sets to which that block is assigned. If each of the blocks in the table is 1k, for example, then the pattern recurs every 4k bytes and the entire stripe, including parity, may be computed in memory by buffering 15 x 4 = 60k bytes of data. In addition, all parity for that stripe may be computed in memory 524 prior to performing one large write operation to the disks.

In each data disk, 4 of the 12 parity sets are not represented. Also, no parity set is represented twice on a disk. Furthermore, no data block belongs to two parity sets for which the parity blocks are stored on the same parity disk. As noted, this is key to the success of the concentrated parity technique, as no more than two members of any parity set can be lost with just two disk failures and at least 4 parity sets will always lose only one member. From there, it may be demonstrated that the loss of any pair of disks can be recovered by sequentially reconstructing data blocks as missing members of each parity set are filled.

As an example of a reconstruction sequence, assume that disks 0 and 1 are lost to failure. Two members of parity sets 2, 3, 4, 9, 10 and 11 are lost and one member of parity sets 0, 1, 5 and 8 is lost. Blocks B1,9 and B8,10 from disk 0 can immediately be reconstructed, as can block B5,0 from disk 1. Completing the reconstruction of block B1,9 allows the reconstruction of block B9,11, and then block B4,11. Completing block B8,10 allows reconstruction of block B2,10, then block B2,3, then block B3,4, thereby completing the reconstruction of the lost data. Note that the parity deltas among all the rows are unique. This is a general restriction on the overall solution to the concentrated parity technique. Note also that in this parity assignment, there are no parity deltas of Δ3, Δ6 or Δ9 in any row of data blocks. This, in turn, allows the grouping of four different parity sets on each parity disk.

It is possible to construct a "balanced" array where all the disks of the array have the same amount of information contained thereon, whether data or parity. For example, referring to the concentrated parity configuration of Fig. 7, each of the 9 disks, (6 data and 3 parity disks) contain the same amount of information (two blocks) for each stripe across the disks. When the stripe is repeated multiple times across the disks, the disks are filled uniformly and, accordingly, all the disks may be of the same size, thereby providing a balanced array. Notably, the array 700 is balanced after one repetition of a stripe.

However, assume that for a 10-disk array, there are 5 blocks per disk per stripe that results in 5 rows of blocks per stripe across the 10 disks of the array. Assume further that the last row of blocks across the 10 disks are parity blocks and that those blocks are removed from the 10 data disks and placed on a disjoint set of 5 parity disks, wherein each parity disk contains two parity blocks. The resulting array comprises 10 data disks, each with 4 blocks per stripe, and 5 parity disks, each with 2 blocks per stripe, that collectively form a first unbalanced parity stripe array.

A second unbalanced parity stripe array having 10 data disks, each with 4 blocks, and 5 parity disks, each with 2 blocks, may be combined with the first unbalanced parity stripe array to thereby form a balanced parity "super-stripe" array. As used herein, an unbalanced parity stripe array may comprise an arrangement wherein the stripe depth is not uniform among the disks such that there are fewer parity blocks per disk than data blocks per disk in the stripe. However, when the number of parity blocks per disk divides evenly into the number of data blocks per disk in a stripe, a plurality of unbalanced parity stripe "arrays" may be combined to form a large, balanced parity "super-stripe" array. The resulting array has separate sets of data disks and another set of parity disks containing parity that is "overlapped" for all of the data disks.

Fig. 9 is a block diagram illustrating a balanced parity super-stripe array 900. The resulting balanced super-stripe array comprises 25 disks, 20 of which are data disks and 5 of which are parity disks, wherein each disk has 4 blocks per stripe. Thus, if the depth of the parity disks is some integer fraction of the depth of the data disks, the parity information from two or more different disk arrays can be combined onto one set of the parity blocks. In an alternate embodiment, the parity from each of the unbalanced parity stripe arrays can be combined onto the same 5 parity disks, interleaving parity from each disk array in groups of two blocks. The entire set of 25 disks may be considered as a single array for large write operation purposes, writing out a large stripe 4 blocks deep by 25 blocks wide with 80 data blocks on 20 data disks and 20 parity blocks on 5 parity disks. Fig. 10 is a block diagram illustrating such a balanced array configuration 1000 adapted for large write operations.

The present invention comprises a parity assignment technique that allows for parity declustering in a large, balanced parity ("super-stripe") array of a storage system. The parity assignment technique is preferably implemented by the storage (RAID) layer 624 and may apply to a distributed parity implementation, although the illustrative embodiment is directed to a concentrated parity implementation. As described herein, the novel parity assignment technique distributes the assignment of disks to parity groups within the array such that all disks contain the same amount of data or parity information.

Moreover, the inventive technique ensures that all surviving data disks of the array are loaded uniformly during a reconstruction phase after a single or double disk failure.

Broadly stated, an aspect of parity declustering involves assigning the data blocks to a parity block of a parity set. For example, assume a RAID-5 implementation having a stripe depth of 5 blocks (4 data blocks and 1 parity block). As more disks are added to the disk array, the same ratio of 4 data blocks to 1 parity block per parity set is maintained for each disk. The advantage of parity declustering is that when a disk is lost, all the data does not have to be read from all of the disks in order to reconstruct that failed disk. This is because the data is lost from a particular parity set that is not dependent upon all the data on all disks in the extended array. Assume further that the disk array originally included 5 disks and is thereafter extended to 9 disks. When one disk fails, data from only 4/8 or 1/2 of the total number of blocks must be accessed in order to reconstruct the failed disk. In other words, for every block that is lost in the extended array, 4 other blocks must be read from the remaining 8 disks in order to recover (reconstruct) the lost block. If one disk is lost, the inventive technique provides for optimal declustering, whereas if 2 disks are lost generally all disks must be accessed to recover the lost data.

According to the present invention, parity declustering can be achieved by selecting patterns of characters, such as binary numbers, representing the data block disks constituting a stripe to thereby change the association of the data disks with parity groups from stripe to stripe of the balanced super-stripe array. In this context, parity declustering refers to distributing the association of disks to parity groups more widely than initially configured so as to distribute the reconstruction workload across all surviving disks as much as possible. In other words, the parity assignment technique changes the parity group association of data blocks of a data disk with respect to other data blocks of data disks of the array. Thereafter, in the event of a single or double disk failure, the reconstruction load may be distributed more uniformly across all of the surviving data disks.

Fig. 11 is a schematic block diagram illustrating the parity assignment technique of the present invention as applied to a large balanced array 1100 constructed in accordance with Figs. 9 and 10. The balanced "super-stripe" array 1100 comprises 25 disks, 20 of which are data disks and 5 of which are parity disks. Declustering can be achieved by changing the grouping of the 20 data disks into two groups of 10 data disks in each occurrence of the parity assignment pattern. For example, assume one parity group is "red" (R) and the other is "green" (G). The data (blocks) disks of the array are organized such that different blocks are logically participating in different parity sets from stripe to stripe across the array. In this context, parity declustering involves adding disks to a disk array without increasing the parity set size in such a way that the blocks associated with the parity sets are distributed as uniformly as possible across the rest of the disks of the array.

Fig. 12 is a diagram illustrating an alternate assignment of data disks to parity groups in a repeating pattern 1200 of R and G characters. Note that the mathematical combinatorics notation $\begin{bmatrix} a \\ b \end{bmatrix}$ (i.e., choose b elements out of the set of a) may be advantageously employed to enumerate all combinations of 10 R and 10 G patterns representing the uniform distribution of the reconstruction workload. That is, all $\begin{bmatrix} 20 \\ 10 \end{bmatrix}$ patterns can be enumerated, with repetition occurring only when all of the patterns are exhausted.

When one data disk fails, less than half of the parity data and slightly less than half of the actual data must be retrieved from the surviving disks in order to reconstruct the contents of the failed disk. In fact, only 46% of the total data and parity must be retrieved from the disks. This speeds up the reconstruction process and allows greater availability of the array during reconstruction. When two disks fail, most of the parity data must be retrieved, but slightly less than 3/4 of the data must be retrieved from each data disk. Actually, only 67% of the total data and parity must be accessed from the disks in order to recover all of the lost data from two disk failures.

In sum, for a single data disk failure, each data disk is accessed for 9/20 of its blocks and each parity disk is accessed for half of its blocks. For a double data disk failure, each surviving data disk is accessed for 14/19 of its blocks and each parity disk is accessed for 29/38 of its blocks.

According to an aspect of the present invention, a canonical ordering of the assignment patterns can be easily defined. If, instead of “R” and “G” characters, binary digits (e.g., 0 and 1) are used, then the canonical order may be defined as simply being the sequential order of the patterns interpreted as, e.g., 20-bit numbers. Translation from an ordinal position to a 20-bit pattern may be achieved either by using a look-up table or software program code. The program code can be made faster by pre-computing a look-up table for all the required $\begin{bmatrix} a \\ b \end{bmatrix}$ combinations in a two dimensional array.

Fig. 13 is a diagram illustrating an assignment pattern 1300 of data (block) disks to parity groups using binary digits 0 and 1 for a disk array having 8 data disks. Each row or stripe of the assignment pattern is considered an 8-bit binary number, such that the first stripe has a value of 15. The total number of 8-bit numbers having four 0s and four 1s may be determined by the notation $\begin{bmatrix} 8 \\ 4 \end{bmatrix}$. The notation $\begin{bmatrix} 8 \\ 4 \end{bmatrix}$ is equal to 8!/4!4! or a total of 70 different combinations of four 1s and four 0s. If the 8-bit numbers are placed in order based on their unsigned binary values, a canonical ordering of these numbers is provided. Thus, although there are 256 different 8-bit numbers, only 70 of those different numbers are relevant according to the $\begin{bmatrix} a \\ b \end{bmatrix}$ notation. Moreover, those 70 numbers are sorted into order from lowest value to highest value.

Specifically, in the first row or stripe, the first 4 data disks are associated with a first parity group and the last 4 data disks are associated with a second parity group. In the second row or stripe, the first 3 disks and the 5th disk are associated with the first parity group, whereas the 4th disk and the last 3 disks are associated with the second parity group. In the third stripe, the first 3 disks and the 6th disk are associated with the first parity group, while the 4th and 5th disks and the last 2 disks are associated with the second parity group. The sequence continues until the last stripe where the first 4 disks are associated with the second parity group while the last 4 disks are associated with the first parity group. Since the intent is to "mix up" the associations of the data disks with the parity groups, there really is no difference between the pattern in the first and last stripes and, accordingly, there are only 35 relevant 8-bit numbers. These latter relevant numbers consist of all 8-bit numbers that have four 1s asserted with a zero in the most significant bit.

For example, refer to the 8th stripe of the parity assignment pattern 1300. Here, the parity generation pattern for parity group 1 is applied to the 1st, 2nd, 4th and 7th data disks, while the parity generation pattern for the second parity group is applied to the 3rd, 5th, 6th and 8th data disks. As noted, declustering involves distributing the assignment of data blocks to parity sets to thereby distribute the workload during reconstruction. Therefore, the assignment of data blocks to parity sets changes from stripe to stripe as indicated by the difference between the 8th stripe and the 1st stripe of parity assignment pattern 1300.

Thus, if the 1st disk of the 8th stripe is lost, data must be retrieved from the 2nd, 4th and 7th disks in order to reconstruct the lost data. However, in the 9th stripe, data is retrieved from a different combination of three data disks in order to reconstruct the lost data disk. In general, there are three data disks out of the total of seven remaining data disks that must be accessed per stripe in order to reconstruct the lost (failed) data disk. Therefore, on average, 3/7 of each surviving data disk must be read (accessed) in order to reconstruct a lost disk of the 8 data disk array.

Fig. 14 is a block diagram of an array 1400 having 6 data disks (Disks0-5) assigned to parity groups using binary digits 0 and 1 in accordance with a parity assignment pattern using the inventive technique. The assignment pattern is defined in canonical ordering as a sequential order of patterns interpreted as 6-bit numbers which, according to the $\begin{bmatrix} 6 \\ 3 \end{bmatrix}$ notation, results in 10 unique patterns of binary digits 0 and 1. Assume the 2nd disk (Disk1) fails, as represented by the "loss" of the second column of array 1400. It is desired to determine the extent of distribution of the reconstruction workload across all the surviving disks (as defined by the parity declustering technique).

To that end, a determination is made with respect to the number of times each surviving disk is accessed ("hit") when reconstructing the lost data across each stripe (row) of the disk array. This determination is made by examining the particular parity set associated with the lost data per stripe of the array, i.e., the parity set or binary state associated with the lost data across each stripe. Using this method, it can be shown that each of the surviving disks is hit 4 times when reconstructing the lost data associated with failed disk 1. This is an illustrative example of an optimal parity declustering arrangement. In other words, each of the surviving 5 disks are hit 40% of the time, rather than 100% of the time for some of the disks and 0% of the time for the remaining disks, during reconstruction of the failed disk.

While there has been shown and described an illustrative embodiment for parity declustering in a large, balanced, parity super-stripe array of a storage system, it is to be understood that various other adaptations and modifications may be made within the spirit and scope of the invention. For example, any number of disk arrays may be combined to enable parity declustering in large balanced disk arrays. In yet another alternate embodiment of the invention that combines three disk arrays, the declustering patterns are enumerated by weighting all combinations of equal numbers of the digits 0, 1 and 2 into ternary numbers. It can be shown that three arrays, each having two disks, can be combined into a 6-disk balanced array wherein the association of the data disks to parity groups may be "juggled" from stripe to stripe. By juggling this association, the number of times each surviving disk is accessed (hit) during reconstruction of a lost disk is equal and balanced across the surviving disks of the array. Fig. 15 is a diagram showing a partial parity assignment pattern 1500 for a balanced array comprising three arrays, each having two disks. In response to a disk failure, a balanced impact on the surviving disks of the array arises, resulting in less than 1/3 of the data on each disk being retrieved during reconstruction of the lost disk.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is: